

SAT/SMT ソルバのしくみ

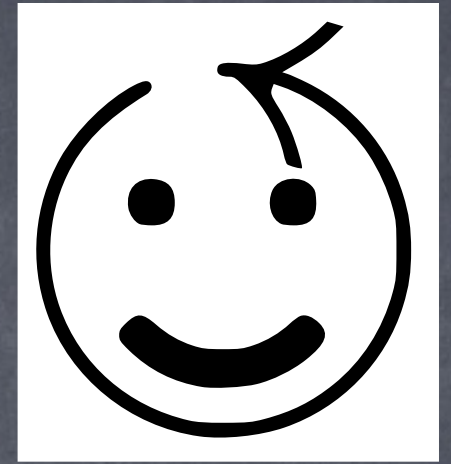
酒井 政裕

2015-09-12

Proof Summit 2015

改訂版

自己紹介: 酒井 政裕



- Twitter: @masahiro_sakai
- github: <https://github.com/msakai/>
- G+: <https://plus.google.com/+MasahiroSakai>
- ブログ: ヒビルテ
- Haskell
- 共訳書に Alloy本 や TaPL
- 定理証明や決定手続きに興味
 - <https://github.com/msakai/toysolver>
 - 今日はその中から、SAT/SMTソルバについて紹介

CM: Alloy本&TaPL好評発売中

抽象によるソフトウェア設計 Alloy ではじめる形式手法

Daniel Jackson 著

中島 豊 監訳

今井 健男・酒井 政裕・遠藤 佑介・片岡 欣夫 共訳

Software Abstractions
Logic, Language, and Analysis



型システム入門 プログラミング言語と型の理論

Benjamin C. Pierce 著

Types and Programming Languages

住井英二郎 監訳

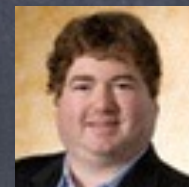
遠藤佑介・酒井政裕・今井敬吾・黒木裕介・今井宣洋・才川隆文・今井健男 共訳



Courserian

最近はCourseraでデータ分析・機械学習に
手を出してみたりも。

- ◉ Computing for Data Analysis
 - ◉ by Roger D. Peng @ Johns Hopkins University
 - ◉ Data Analysis
 - ◉ by Jeff Leek @ Johns Hopkins University
 - ◉ Machine Learning
 - ◉ by Andrew Ng @ Stanford University
 - ◉ Introduction to Recommender Systems
 - ◉ by Joseph A Konstan and Michael D Ekstrand @ University of Minnesota
 - ◉ Process Mining: Data science in Action
 - ◉ by Wil van der Aalst @ Eindhoven University of Technology
- などなど



朝日新聞 2013.8.22 教育欄

「ネット講義 東大を世界に」



NHKクローズアップ現代2013.09.17

「あなたもハーバード大へ
～広がる無料オンライン講座～」



『ルポ MOOC革命』

金成 隆一

本日のお題 (1) : SAT

- SAT = SATisfiability Problem
 - 論理式の充足可能性問題
 - 論理変数を含む命題論理式が与えられたとき、
「それを真にする(=充足する)割り当て」があるか?
 - 例: $(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee \neg Q)$
 - 充足可能 ($P = \text{true}, Q = \text{false}$ のときに真)
 - 例: $(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee \neg Q) \wedge (\neg P \vee Q)$
 - 充足不能
- SATソルバは命題論理式の充足可能性を判定
 - 充足可能な場合には割り当ても出力

本日のお題 (2) : SMT

- SMT = Satisfiability Modulo Theories

- 例:

- $0 \leq b \wedge b < 10 \wedge$

- $(2 \times b + 1 = c \vee \text{read}(a, b) = 0) \wedge$

- $f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1)$

- を真にする a, b, c, f の解釈は存在するか?

- SATが命題論理なのに対して、SMTは述語論理の豊かな表現力

- SATソルバの次としてソフトウェア検証などで普及

- 詳細は後述

特徴: 速くて便利

- SATは古典的なNP完全問題だが、近年SATソルバは著しく高性能化
- 様々な問題を、SAT/SMT問題へとエンコードし、ソルバを用いて解くアプローチが普及
 - ソフトウェア・ハードウェア検証、プランニング、...
 - 利点
 - ソルバの進歩によるメリットを享受
 - 「制約生成」「制約解消」という関心事の分離
⇒ 開発効率の向上&メンテナンス容易性
 - 欠点 (ICFP Programming Contest のオーガナイザの話)
 - "Elegant general-purpose formulations in terms of constraint solving: Relatively easy to code up and obtain decent results"
 - "But, hand-tuned solutions are going to do better ... MUCH BETTER"

特徴: 論理的に面白い

- 証明論やモデル論の基本が応用に直結していて、面白い
 - 証明論的なアプローチとモデル論的なアプローチの組合せ
 - クレイグの補間定理
(Craig's interpolation theorem)
 - レーヴェンハイム-スコーレムの定理
(Löwenheim-Skolem theorem)
 - などなど
- Proof Summit 的にもどう？

充足可能性 vs. 定理証明

論理式の分類

恒真 (valid)

- 任意の割り当てで真: $M \models \phi$ for all M

- 例: $P \vee \neg P$, $3 > 2$

充足可能 (satisfiable)

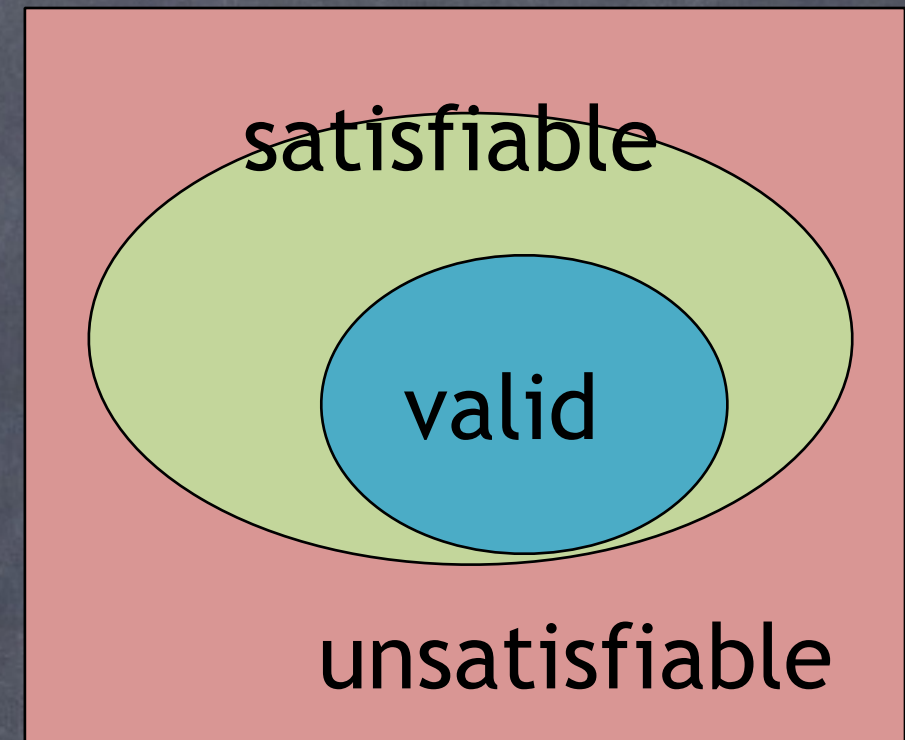
- 真になる割り当てが存在: $M \models \phi$ for some M

- 例: $P \vee \neg Q$, $3 > x$, $P \vee \neg P$

充足不能 (unsatisfiable)

- 真になる割り当てが存在しない: $M \not\models \phi$ for all M

- 例: $P \wedge \neg P$, $x > 2 \wedge 0 \geq x$



定理証明と充足可能性の関係 (完全性より)

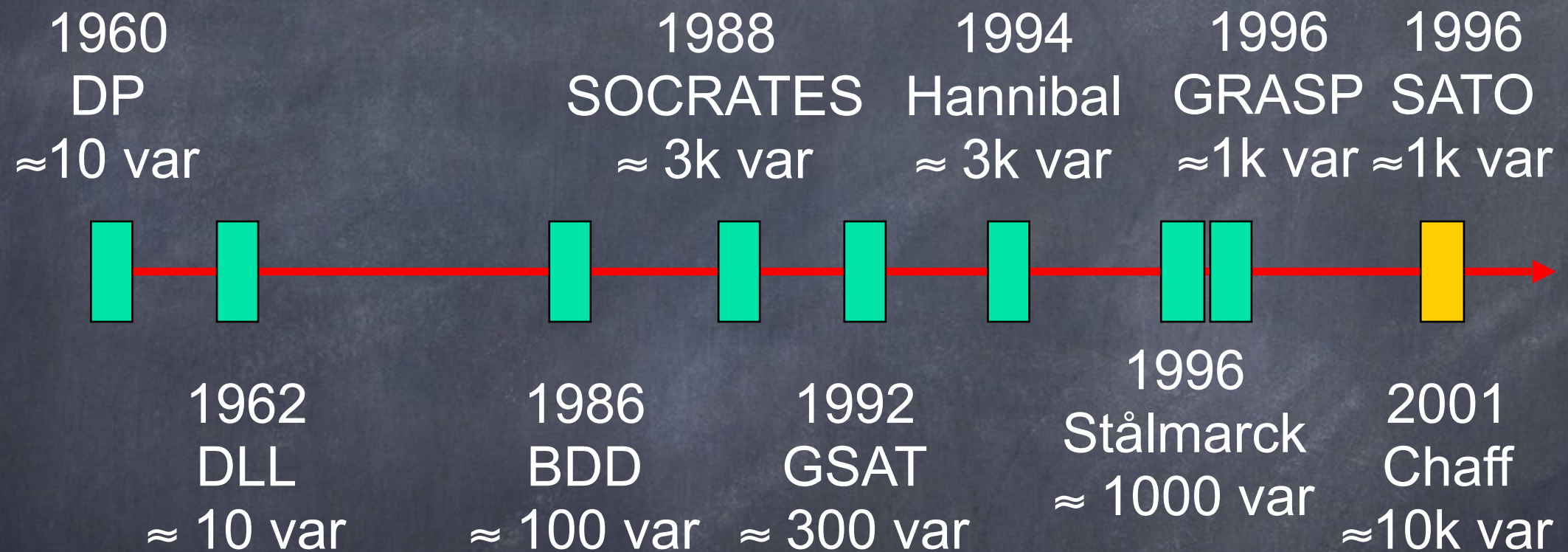
- ϕ が証明可能 $\Leftrightarrow \phi$ が恒真 $\Leftrightarrow (\neg \phi)$ が充足不可能

- すなわち、 ϕ を証明するには $\neg \phi$ が充足不可能なことを示せばよい。

SAT編

SATの性能向上の歴史

B.C. (Before Chaff) 時代



http://www.cs.cmu.edu/~emc/spring06/home_files/zchaff4ed.new.ppt

SATの性能向上の歴史(2)

- 1996 GRASP: ブレイクスルーの始まり (学習の導入)
- 2001 Chaff: 今のソルバの基本形
- 2005 Minisat: オープンソースでコンパクト
 - SAT 2005 competition の3つのindustrial category を総なめ。
 - 以降広く利用 & ソルバ開発のベースラインに
- 2015現在
 - 今の規模感
 - 数百万変数・1千万以上の制約 (industrialな問題)
 - 規模が小さくても難しい問題があるので、一概には言えない

SATの入力

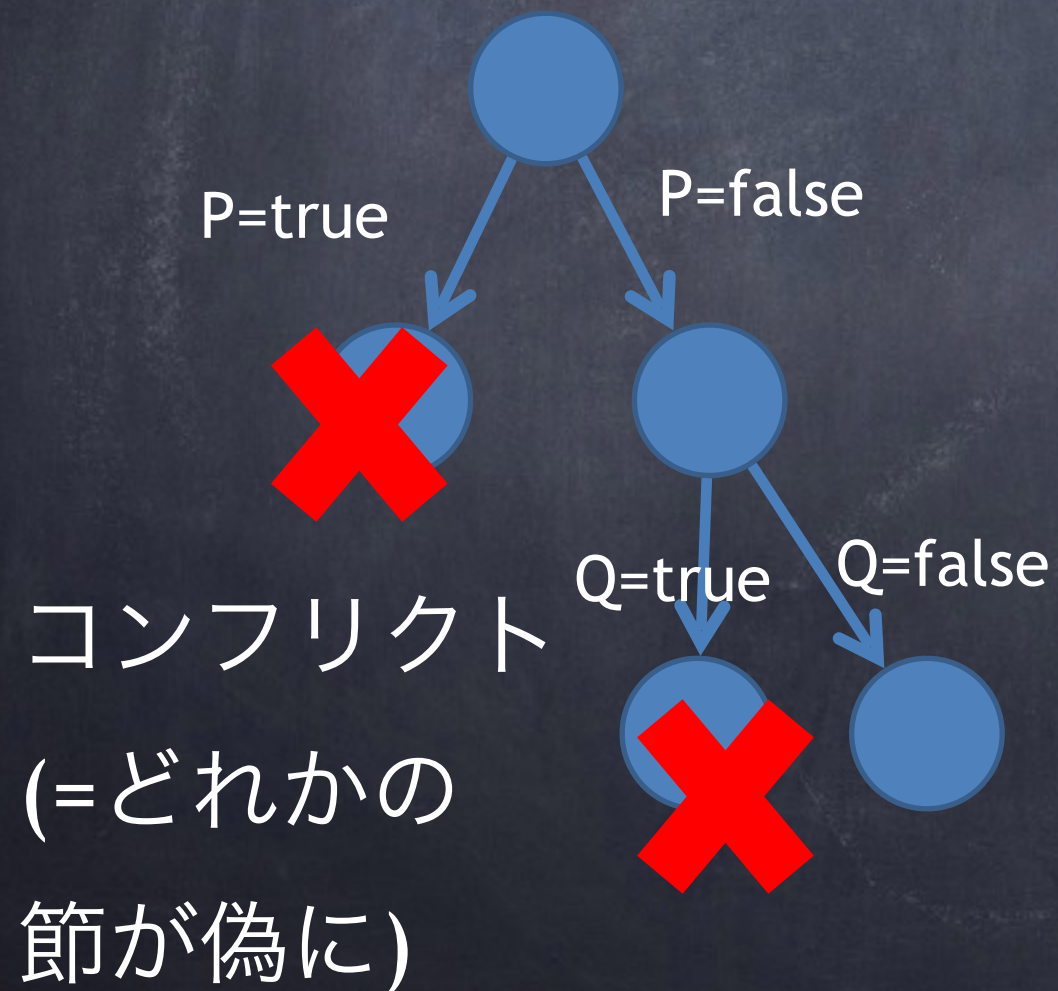
- 普通のSATソルバは任意の論理式ではなく、
CNF形式の論理式を扱う
 - $CNF = \text{Conjunctive Normal Form}$
 - リテラル ::= 変数 $\mid$ $\neg$ 変数
 - 節 ::= リテラル $\vee \dots \vee$ リテラル
 - $CNF ::= \text{節} \wedge \dots \wedge \text{節}$
- ただし、任意の論理式は充足可能性が同値(equi-satisfiable)
な線形サイズのCNFへと変換可能 [Tseitin68]

Tseitin encoding

- 部分式を変数に置き換え、それが元の部分式と同値であるという制約を追加
- 例)
 - $\phi[a \wedge b]$
 - $\Rightarrow \phi[c] \wedge (c \leftrightarrow a \wedge b) \dots c$ はフレッシュな変数
 - $\Rightarrow \phi[c] \wedge \neg c \vee a \wedge \neg c \vee b \wedge c \vee \neg a \vee \neg b$
- このような置き換えを再帰的に繰り返すことによって、任意の論理式をCNFに変換可能

SATのアルゴリズム

素朴なアイデア:
全組み合わせを列挙
(バックトラッキング)



制約伝播 (Unit Propagation)

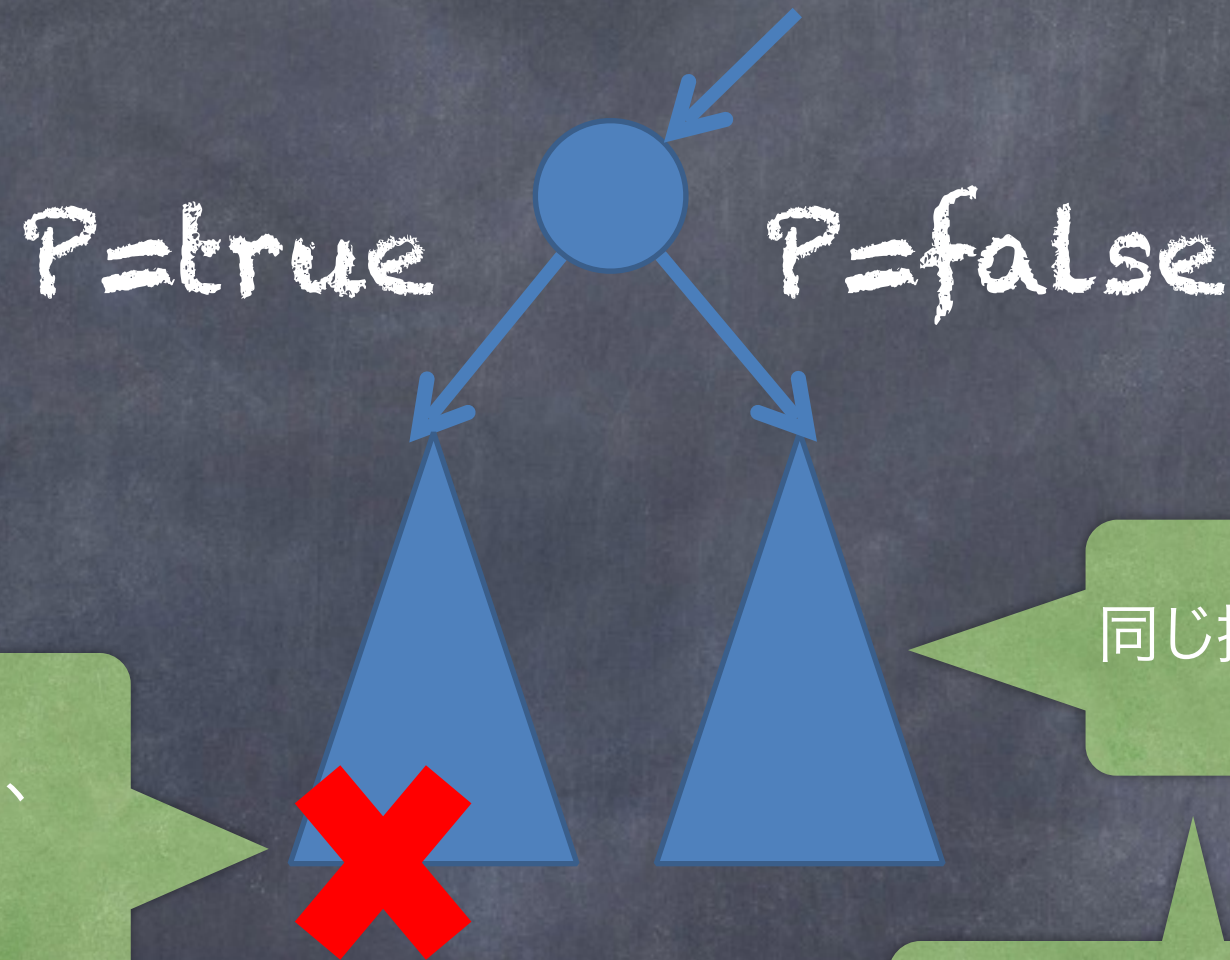
割り当ての結果、節の一つ以外のリテラルが全て偽になったら、最後のリテラルを真にするよう割り当てる

例:

$P \vee \neg Q \vee \neg R$ という節があり、
 $P=false, R=true$ という割り当てが
なされたら、 Q に $false$ を割り当てる

$M \models \phi$ なる M を探す
モデル論的アプローチ

SATのアルゴリズム：学習の必要性



長い探索の果てに、
Pとは無関係に
充足不能なことが判明

同じ探索を再度繰り返す？

Pの値を変えても充足不能
なことは明らか。
学習と制約伝播により回避できる

Those who do not learn from their
mistakes are doomed to repeat them

矛盾からの学習

Conflict-driven Clause Learning (CDCL)

学習

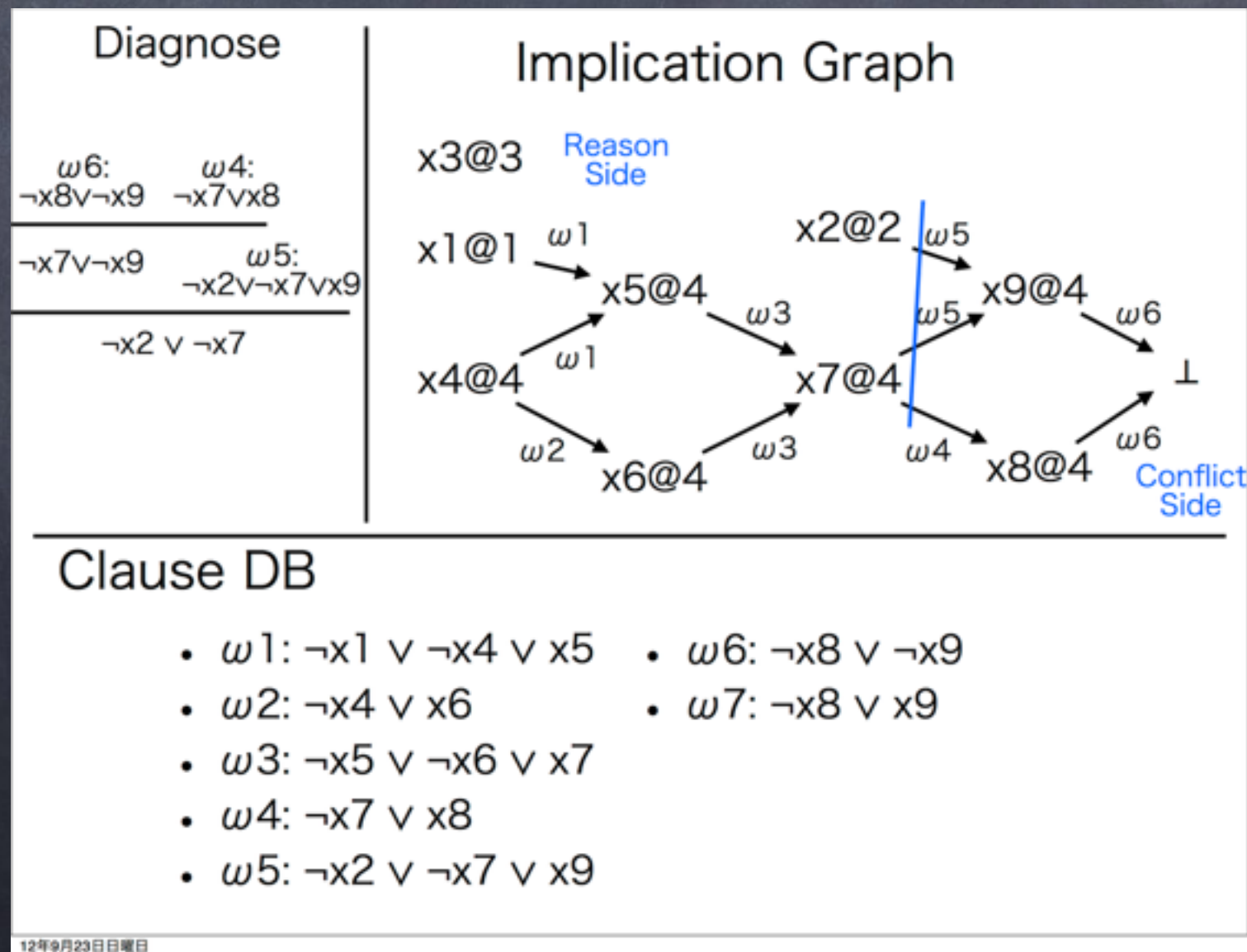
- 部分問題が充足不能と判明したとき、その原因と同じ割り当てをしない制約(学習節)を追加 (制約伝播をどこまで遡るか等の方式は色々)
- 例: 割り当て $P=true, Q=false, R=false$ が原因なら $\neg P \vee Q \vee R$
- 論理的には導出原理(resolution)による推論で、
学習節 C は元の論理式から含意される ($\phi \vdash C$)
- 非時間的バックトラック (Non-chronological backtracking)
 - 直前ではなく、コンフリクト原因を修正できるところまでバックトラック
- 最終的に空節が導出された($\phi \vdash \perp$)ら、 ϕ は充足不能
 - 学習の仮定を記録しておけば、 $\neg\phi$ の証明も生成可能

証明論的

CDCLの動作の様子

- How a CDCL SAT solver works

- <http://www.slideshare.net/sakai/how-a-cdcl-sat-solver-works>



その他の要素の一部 (詳細略)

- 制約伝播のための効率的なデータ構造
 - あるリテラルが偽になった際に、それが出現するすべての節について、制約伝播可能か調べるのは大変
- 変数の決定順ヒューリスティクス
 - 最近のコンフリクトに多く寄与したリテラルを重要なリテラルと判断など
- 学習節の重要性の評価指標
- 割り当てのキャッシュ
- リスタート (学習節や変数の重要性などは保持)
- **CNF**からの構造の復元：論理回路、**XOR**制約、基数制約
- **etc.**

代表的なソルバ

- ◎ 良く使われるオープンソースのソルバ
 - ◎ Minisat (C++), picosat (C), SAT4J (Java)
 - ◎ アプリ組み込みにはこれらを使うことが多い
 - ◎ 性能的にはベストではないが、使い勝手は良い
 - ◎ 試したこと無いけど、Google Optimization Tools にも
SATソルバが入ったっぽい？
- ◎ 最近の競技会で好成績のソルバ
 - ◎ Lingeling, glueminisat, CryptoMiniSat, ...
- ◎ おまけ
 - ◎ 私が学習用に書いた pure Haskell な SATソルバ toysat
(拙作のtoysolverパッケージに入ってる)

デモ: toysat

↓ 入力ファイル

```
p cnf 250 1065
-159 -234 197 0
-71 13 194 0
45 218 38 0
191 -129 -88 0
117 -164 -29 0
1
1
-
25 -51 -165 0
247 31 -64 0
156 228 11 0
64 199 -162 0
1 173 -54 0
136 -98 -215 0
201 -206 159 0
-223 -137 7 0
-34 220 236 0
-155 -92 217 0
...
```

$\neg x_{159} \vee \neg x_{234} \vee x_{197}$

実行例

```
$ toysat UF250.1065.100/uf250-01.cnf
```

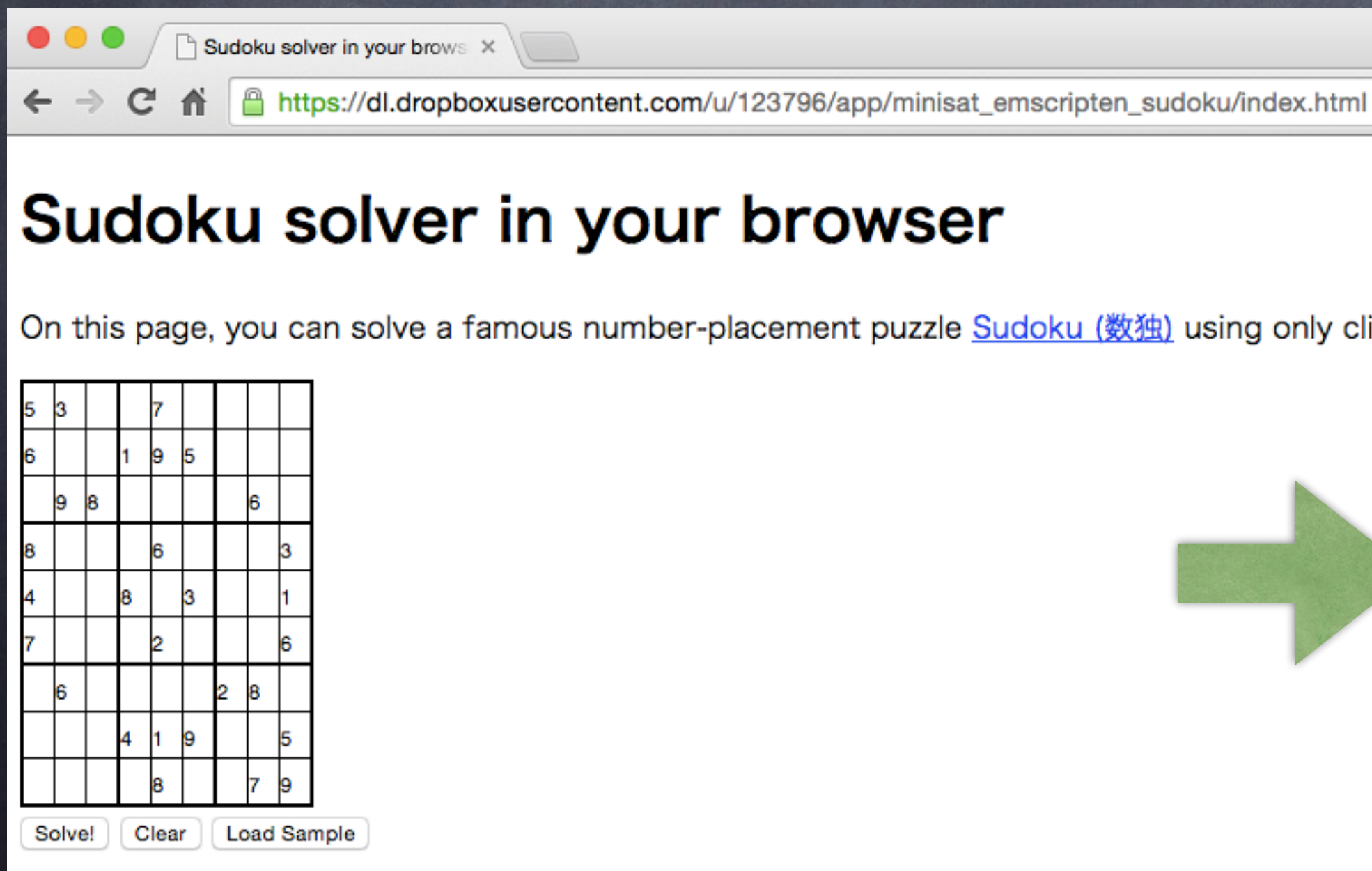
```
...
c #vars 250
c #constraints 1065
c Solving starts ...
c =====[ Search Statistics ]=====
c  Time | Restart | Decision | Conflict |      LEARNT      | Fixed | Removed
c      |        |          |          |      Limit      GC | Var   | Constr
c =====
c 0.0s |      0 |      0 |      0 |      438      0 |      0 |
c 0.0s |      1 |     155 |     100 |      438      0 |      0 |
c 0.1s |      2 |     343 |     250 |      482      0 |      0 |
c 0.1s |      3 |     602 |     475 |      531      0 |      0 |
c 0.1s |      4 |    1014 |     813 |      585      1 |      0 |
c 0.2s |      5 |    1622 |    1320 |      644      3 |      0 |
c 0.4s |      6 |    2560 |    2081 |      780      4 |      0 |
c 0.6s |      7 |    3927 |    3223 |      859      7 |      0 |
c 1.1s |      8 |    5948 |    4936 |      945     11 |      0 |
c 1.6s |      9 |    9039 |    7506 |     1040     16 |      0 |
c 2.3s |     10 |   13555 |   11361 |     1144     23 |      0 |
c 3.3s |     10 |   19494 |   16361 |     1144     32 |      0 |
c 3.4s |     11 |   20419 |   17144 |     1259     33 |      0 |
c 4.0s |     11 |   23520 |   19665 |     1259     37 |      0 |
c #cpu_time = 3.872s
c #wall_clock_time = 3.957s
c #decision = 23520
c #random_decision = 105
c #conflict = 19665
c #restart = 11
s SATISFIABLE
v -1 -2 3 -4 5 -6 7 -8 9 -10
v -11 -12 -13 14 15 16 17 -18 19 20
v 21 22 -23 -24 25 26 -27 28 29 -30
v 31 32 -33 34 -35 36 37 -38 39 -40
...
```

充足可能

$x_1 = \text{false}, x_2 = \text{false},$
 $x_3 = \text{true}, \dots$

デモ: ブラウザ上でMinisatで「数独」を解いてみる
(EmscriptenでJavascriptにコンパイル)

https://dl.dropboxusercontent.com/u/123796/app/minisat_emscripten_sudoku/index.html



Sudoku solver in your browser

On this page, you can solve a famous number-placement puzzle [Sudoku \(数独\)](#) using only cli

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Solve! Clear Load Sample

Result

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Satisfiability Modulo Theories (SMT)

基本編

SATの弱点 $\Rightarrow$ SMTソルバ

● SATの欠点

- 問題によっては、SATにエンコードすると.....
- 規模が大きくなりすぎる
- 抽象度が低くなって、元の問題にあった構造が活用できなくなる

● SATソルバを専用ソルバ（例えば算術演算に関するソルバ）と組み合わせて、弱点を克服

● $\Rightarrow$ SMTソルバ

SMT = SATソルバ + 理論ソルバ

- 発想

- SATソルバを専用ソルバ（例えば算術演算に関するソルバ）と組み合わせて、弱点を克服

- 構成

- 論理的な構造はSATが担当
- 背景理論に関する推論(例: $a \geq 2 \Rightarrow a \neq 0$)はその理論のソルバが担当

SMTの動作: オフライン版

$a > 0 \vee b - 2c \geq 0,$
 $\neg(a > 0) \vee b = c,$
 $\neg(a > 0) \vee c \geq a,$
 $b < c \vee c \geq a$

Boolean
abstraction

$P \vee Q,$
 $\neg P \vee R,$
 $\neg P \vee S,$
 $T \vee S$

制約追加

理論
ソルバ

UNSAT

$\{a > 0, b -$
 $2c \geq 0,$
 $b = c, c \geq a\}$
が矛盾

$\neg P \vee \neg Q$
 $\vee \neg R \vee \neg S$

SATソルバ

SAT

SAT

UNSAT

$a > 0 \wedge b - 2c \geq 0$
 $\wedge b = c \wedge c \geq a$
 $\wedge \neg(b < c)$

リテラル
の連言

$P = Q = R = S = \text{true}$
 $T = \text{false}$

SMTの動作: DPLL(T)

- 今普及している手法(Lazy SMT) DPLL(T)

- SATソルバと理論ソルバが「オンライン」で連携

- SATソルバ $\Rightarrow$ 理論ソルバ

- SATソルバが終了してから理論ソルバを呼ぶのではなく、
- 命題変数に真偽を割り当てるたびに、理論ソルバ側に制約を追加。
- 適当なタイミングで整合性を検査。
- バックトラックする際に、理論ソルバ側から制約を削除。

- 理論ソルバ $\Rightarrow$ SATソルバ

- 理論ソルバが矛盾を検出した時点で枝刈り & 学習
- 含意される原子論理式の真偽の割り当て
 - 例: $a = 0, a > 0$ という原子論理式がそれぞれ P, Q という論理変数で表されているとして、 P に **true** が割り当てられたときに、 Q に **false** を割り当てる。

(改めて) SAT問題

- (量化子を含まない) 一階述語論理式が対象
 - SATの命題論理に対して、豊かな表現力
- 背景理論(theory)のもとでの充足可能性
 - 理論:
 - 「公理系」 「推論に関して閉じた論理式集合」 「モデルのクラス」 (どの定義を用いるかは流儀による)
 - 実際的には、一部の述語/関数/定数(例: $\leq$, $+$, `read`, 3.0)の解釈が固定されていると思えば良い
 - 固有のソルバを使って高速に推論

自動定理証明器との違い

● SMT

- 一部述語・関数(e.g. $\leq$, $+$, $read$)の解釈が背景理論によって固定
- 固定された解釈に基づいた高速な専用アルゴリズムを活用
- 量子子への対応は限定的
- 決定可能な場合を扱い、高速
- 一階述語論理の(古典的な)自動定理証明
 - 解釈は固定されていない
 - そのためすべてを明示的に公理化する必要
 - 量子子も自由に扱え、汎用度は高い
 - 一般には半決定可能でしかなく、低速

例: ペアノ算術

- $\forall x. 0 \neq s(x)$

- $\forall x1, x2.$
 $s(x1) = s(x2)$
 $\Rightarrow x1 = x2,$

- $\forall a, b.$
 $a + s(b) = s(a + b)$

- ...

SMTで使われる理論

- 任意に解釈可能な関数と等号に関する理論（公理が空集合）
- 整数や有理数上の（線形 | 非線形）算術
- 配列の理論
 - ソフトウェア・ハードウェア検証でのメモリのモデル化など
- ビットベクタの理論
 - ソフトウェア・ハードウェア検証での整数型のモデル化など
- （余）帰納的データ型の理論（ $\text{List} ::= \text{Nil} \mid \text{Cons } A \text{ List}$ ）
- 文字列の理論
 - 言語処理系向けのテストケース生成、SQLインジェクション脆弱性探しなど
- 浮動小数点数の理論

代表的な理論: EUF

- EUF (Equality and Uninterpreted Function)

- 任意に解釈してよい関数と等号に関する理論
- 変数もゼロ引数の関数記号(定数記号)として考える
- 公理無しが一番基本的な「理論」

- 例:

- $f(b) = c \wedge f(c) = a \wedge g(a) = h(a, a) \wedge \neg(g(b) = h(c, b))$

は充足可能?

- これに $b = c$ を追加した場合はどうか?

- ソルバのアルゴリズム:

- Union-Find を拡張したアルゴリズムで Congruence Closure を計算するなど

```

(set-logic QF_UF)
(set-option :produce-models
true)
(declare-sort U 0)
(declare-fun f (U) U)
(declare-fun g (U) U)
(declare-fun h (U U) U)
(declare-fun a () U)
(declare-fun b () U)
(declare-fun c () U)

(assert (= (f b) c))
(assert (= (f c) a))
(assert (= (g b) (h a a)))
(assert (not (= (g b) (h c b))))
(check-sat)
(get-model)

(assert (= b c))
(check-sat)

```

← 入力ファイル

↓ 実行例

```

$ cvc4 --incremental test.smt2
sat
(model
; cardinality of U is 5
(declare-sort U 0)
; rep: @uc_U_0
; rep: @uc_U_1
; rep: @uc_U_2
; rep: @uc_U_3
; rep: @uc_U_4
(define-fun f ((_ufmt_1 U)) U
  (ite (= @uc_U_2 _ufmt_1) @uc_U_0 @uc_U_2))
(define-fun g ((_ufmt_1 U)) U @uc_U_3)
(define-fun h ((_ufmt_1 U) (_ufmt_2 U)) U
  (ite (= @uc_U_0 _ufmt_1)
    (ite (= @uc_U_0 _ufmt_2) @uc_U_3 @uc_U_4)
    @uc_U_4))
(define-fun a () U @uc_U_0)
(define-fun b () U @uc_U_1)
(define-fun c () U @uc_U_2)
)
unsat

```

代表的な理論: 算術

◉ Linear Real Arithmetics (LRA)

- ◉ 実数(有理数)上の線形算術 ($\equiv$ 線形計画 w/o 最適化)
- ◉ アルゴリズム例: 単体法, Fourier-Motzkin, Virtual Substitution

◉ Linear Integer Arithmetics (LIA)

- ◉ 整数上の線形算術 ($\equiv$ 整数線形計画 w/o 最適化)
- ◉ アルゴリズム例: 単体法+B&B+切除平面, Omega test, Cooper

◉ Non-Linear Real Arithmetics (NRA)

- ◉ 実数上の一般の算術 ($\equiv$ 多項式計画 w/o 最適化)
- ◉ アルゴリズム例: Cylindrical Algebraic Decomposition, Virtual Substitution, Gröbner Basis

◉ Non-Linear Integer Arithmetics (NIA)

- ◉ 整数上の一般の算術 ($\equiv$ ディオファントス方程式)
- ◉ 一般には決定不能[Matiyasevich70]

```

(set-logic QF_AUFLIRA)
(set-option :produce-models true)
(declare-fun x1 () Real)
(declare-fun x2 () Real)
(declare-fun x3 () Real)
(declare-fun x4 () Int)
(assert (<= (+ (- x1) x2 x3 (* 10.0
x4)) 20.0))
(assert (<= (+ x1 (- (* 3.0 x2)) x3)
30.0))
(assert (= (+ x2 (- (* 3.5 x4))) 0.0))
(assert (<= 0.0 x1))
(assert (<= x1 40.0))
(assert (<= 0.0 x2))
(assert (<= 0.0 x3))
(assert (<= 2 x4))
(assert (<= x4 3))
(check-sat)
(get-model)

```

← 入力ファイル

↓ 実行例

```

$ cvc4 --incremental \
  test2.smt2
sat
(model
(define-fun x1 () Real 7)
(define-fun x2 () Real 7)
(define-fun x3 () Real 0)
(define-fun x4 () Int 2)
)
$

```

理論ソルバ

基本的には既存アルゴリズムを利用するが、SMT特有の要件もある

1. インクリメンタルに制約を追加可能
2. 制約の追加前の状態に(低コストで)戻せる
 - SATソルバのバックトラック時の処理
3. $\wedge C$ の矛盾時に、出来るだけ小さな原因(= C の矛盾する部分集合)を発見可能
 - 小さい方がSAT側の制約伝播・枝刈りに効率的
4. 含意される制約を(低コストで)発見可能
 - SATソルバへの制約伝播のため
5. インターフェース等号の導出機能 (後述)

主なSATソルバ

- Z3 (Microsoft Research)
 - 性能および機能の双方で独走
 - 以前は商用利用に制限があったが、オープンソース化
- CVC4 (NYU and U Iowa)
 - オープンソースの定番ソルバ。Z3とはまた違う実験的な機能も
- yices (SRI)
 - Z3以前に一時代を築いたソルバ。最近ソースが公開。
- MathSAT ... 定評のあるアカデミックなソルバ
- Boolector ... 配列やビットベクタの理論に特化したソルバ
- OpenSMT ... 開発は止まっているがシンプルな実装で学習に良い？
- (おまけ) toysat
 - SATソルバに理論ソルバ用のフックを追加したところで止まっている

Satisfiability Modulo Theories (SMT)

理論結合編

Theory Combination

- 背景理論 T_1 , T_2 を組み合わせた論理式の充足可能性を判定したい
- 例 (EUF $\cup$ LIA):
 - $2a \geq b + f(g(c)) \wedge f(b) = c \wedge f(c) = a \wedge$
 $g(a) < h(a, a) \wedge g(b) > h(c, b) \wedge b = c$
- T_1 の理論ソルバと、 T_2 の理論ソルバはあるとする
- それらを使って $T_1 \cup T_2$ の理論ソルバを構成できるか?

Theory Combinationの課題

- $(T1 \cup T2)$ -論理式 ϕ が与えられたとする
- まず ϕ を充足可能性同値な $\phi 1 \wedge \phi 2$ に変形
 - $T1$ -論理式 $\phi 1$ 、 $T2$ -論理式 $\phi 2$
 - $\phi 1$ と $\phi 2$ は変数(定数記号)と等号以外の記号を共有しない

注: 本当は「 T -論理式」という言い方はせず、 T のシグニチャ Σ を定義して、

課題

「 Σ -論理式」と呼ぶべきだが、ここでは簡略化して書いている

- $\phi 1, \phi 2$ がそれぞれ $T1$ -充足可能, $T2$ -充足可能であつても、 $\phi 1 \wedge \phi 2$ が $(T1 \cup T2)$ -充足可能とは限らない

Craig interpolation theorem

● クレイグの補間定理

- $\phi 1 \Rightarrow \phi 2$ ならば
- $\phi 1, \phi 2$ に共通する記号しか含まない論理式 ε が存在し
- $\phi 1 \Rightarrow \varepsilon$ かつ $\varepsilon \Rightarrow \phi 2$

● 同値な表現

- $\phi 1 \wedge \phi 2$ が矛盾するなら
- $\phi 1, \phi 2$ に共通する記号しか含まない論理式 ε が存在し
- $\phi 1 \Rightarrow \varepsilon$ かつ $\varepsilon \wedge \phi 2$ が矛盾

● SMTの場合

- $T1, T2$ を等号と変数のみしか共有しないようにしておく
- $\phi 1 \wedge \phi 2$ が矛盾するとき、等号と共通変数だけなる論理式 ε が存在
(量化子は含まれるかも知れないので注意)
- ε に相当する情報を理論ソルバ間で交換すれば良い!

素朴な Theory Combination

- 共通する変数の上の同値関係 R を列挙
- 各 R について
 - $A_R = \bigwedge_{(s,t) \in R} (s=t) \wedge \bigwedge_{(s,t) \notin R} (s \neq t)$ とする
 - $A_R \wedge \phi_1$ と $A_R \wedge \phi_2$ の充足可能性をそれぞれ検査
 - いずれかの R について両方充足可能であれば良い
- 完全性
 - T_1, T_2 が stably-infinite なら充足不能性を検出可能
 - T が stably-infinite とは:
 - T -充足可能な基礎論理式はサイズ無限の T -モデルで充足可能

Stably-infiniteness

- T が stably-infinite:
 - 「 T -充足可能な基礎論理式はサイズ無限の T -モデルで充足可能」
 - モデルのサイズが無限だと仮定しても一般性を失わない
- レーヴェンハイム-スコレムの定理 (Löwenheim-Skolem theorem)
 - 「可算な一階の理論が無限モデルを持つとき、全ての無限濃度 κ についてサイズ κ のモデルを持つ」
- T_1, T_2 が stably infinite ならば,
 - 理論ソルバ間でモデルサイズの一致を仮定しても良い
 - $\Rightarrow$ サイズの情報を交換しないで良い
 - $\Rightarrow$ 量子子の含まれない基礎等号の情報だけ交換すれば良い

Nelson-Oppman Theory Combination

- T1ソルバが ϕ_1 から含意される $\forall_i e_i$ という形の論理式をすべて計算し、T2ソルバに伝える
 - e_i は共通変数に関する等式 (インターフェース等式)
- T2ソルバ $\rightarrow$ T1ソルバも同様
- どちらかが矛盾を検出するか、新たに伝播すべき制約がなくなるまで、相互再帰的に伝え合う
- 問題点
 - 選言が含意されることを検出するのは大変
 - 受け入れ側も、場合分けが必要になって大変
 - というか、ぶっちゃけ無理！

Convexity

- 理論 T が凸(convex)とは:
 - Γ を基礎リテラルの連言として
 - $\Gamma \vdash_T \bigvee e_i$ iff $(\Gamma \vdash_T e_i \text{ for some } i)$
- T が凸な場合には Nelson-Oppan を単純化可能
 - 等号の選言 $\bigvee e_i$ ではなく、等号 e_i だけを伝えれば十分
- 例:
 - EUF, LRA などは凸、LIA, BV, A は非凸
 - LIA では $\Gamma = (y=1 \wedge z=2 \wedge 1 \leq x \leq 2)$ とすると
 $\Gamma \vdash_T (x=y \vee x=z)$ だが
 $\Gamma \vdash_T x=y$ も $\Gamma \vdash_T x=z$ も成り立たない

Theory Combination の実際

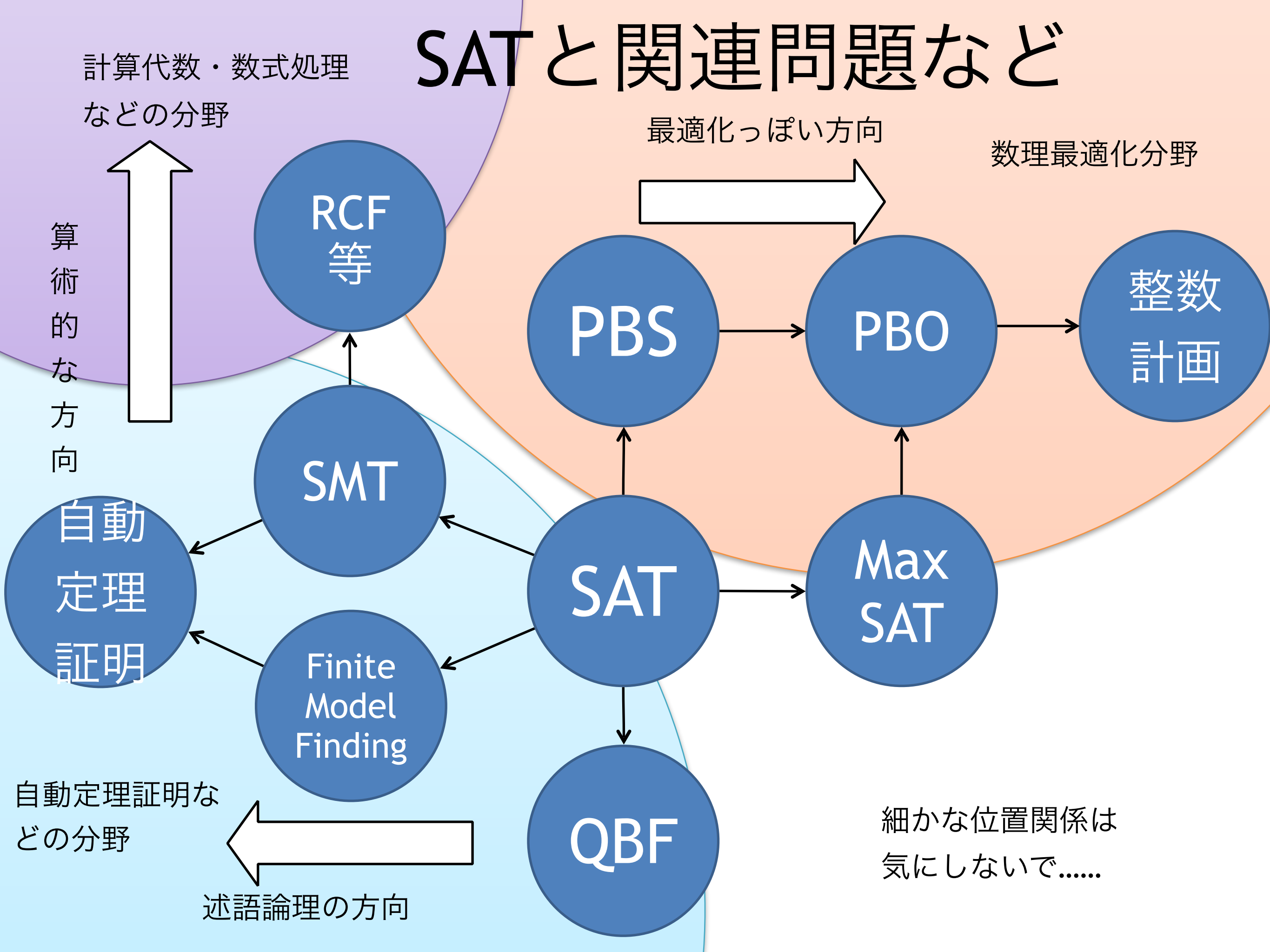
- Delayed Theory Combination (DTC)
 - インターフェース等式を論理変数にして、SATソルバに割り当てを推測させることで、理論ソルバの実装の負担を軽く
- Model-based Theory Combination
 - 単体法のようなモデル生成型の理論ソルバの場合
 - モデルで「たまたま」成り立っている等号も伝播
 - ダメだったらバックトラック
- その他にも理論や実装の研究が色々

まとめとか

まとめとか

- SAT/SMTソルバは高速で便利
- 論理的にも面白い
- 対話的定理証明系とは補完的
 - プログラム検証などでは、VC (Verification Condition) を抽出し、複数のソルバ、自動定理証明器、対話的定理証明器に並列に投げたりも
- 対話的定理証明系内で一部を自動化
 - 例) Isabelle の Sledgehammer

SATと関連問題など



参考文献

- Handbook of Satisfiability
- A. Biere, M. Heule, H. Van Maaren, and T. Walsh, Eds.
- IOS Press, Feb. 2009.
- SAT関係の話は大抵載ってる
- 鈍器としても使用可能



参考文献: 日本語

- <特集>最近のSAT技術の発展, 人工知能学会誌, Vol. 25, No.1 (2010)
 - <http://bach.istc.kobe-u.ac.jp/aisat.html>
- 酒井 政裕, 今井 健男, "SAT問題と他の制約問題との相互発展," コンピュータ・ソフトウェア, vol. 32, no. 1, pp. 103-119, 2015.
 - https://www.jstage.jst.go.jp/article/jssst/32/1/32_1_103/_article/-char/en/

その他 参考

- より「定理証明」っぽいお題をSMTソルバで解いてみた例として
- TPP2011 の TPPmark “Uniform Candy Distribution” をSMTソルバZ3で証明した試み
- <https://github.com/msakai/tpp2011>